

Implementación De Docker En Tres Servicios (Full-Stack) Usando Docker-Compose Y Publicarlo A Docker-Hub.

Contenido

Implementación de docker y su configuración.	3
Prologo	3
Introduccion	5
Descripción	5
Preparación del Entorno de trabajo:	6
1. Estructura del proyecto	9
2. Dockerfile para la base de datos	10
3. Configuración de la API (Spring Boot)	12
4. Configuración de la aplicación (Vue.js)	16
5.-Verificación antes de levantar el docker principal.	20
Estructura Final	24
Subir y Compartir Los repositorios de Docker Hub	25
Conclusión	32
Referencias	33

Implementación de docker y su configuración.

Prologo

“Si tienes una aplicación o servicio y quieres que funcione en diferentes [sistemas como VPS](#) o máquinas, sin ningún problema, considera la posibilidad de utilizar contenedores. Una de las plataformas de contenedores más populares es Docker, aunque no todo el mundo sabe qué es y cómo funciona.

En este tutorial, explicaremos qué es Docker, cómo funciona y en qué se diferencia de las máquinas virtuales (VM) y los sistemas, como Kubernetes y Jenkins. También repasamos los pros y los contras de Docker y enumeramos sus casos de uso más populares”.

¿Qué es Docker?

“Docker es un software de código abierto utilizado para desplegar aplicaciones dentro de contenedores virtuales. La contenerización permite que varias aplicaciones funcionen en diferentes entornos complejos. Por ejemplo, Docker permite ejecutar el sistema de gestión de contenidos WordPress en sistemas Windows, Linux y macOS sin ningún problema”
(Hostinger.com, s.f.)

Algún otro concepto,

(Docker.com, s.f.) “es **Acelerar la forma de crear, compartir y ejecutar aplicaciones**

Docker ayuda a los desarrolladores a crear, compartir, ejecutar y verificar aplicaciones en cualquier lugar, sin tediosas configuraciones o administración del entorno”

Conceptos básicos acerca de Docker que debemos conocer:

1. **Contenedores Docker:** Son entornos aislados y ligeros que contienen todo lo necesario para ejecutar una aplicación, incluidas las dependencias y la configuración. Los contenedores son portátiles y pueden ejecutarse en cualquier máquina que tenga Docker instalado, independientemente de las diferencias en el sistema operativo o el hardware subyacente.
2. **Dockerfile:** Es un archivo de texto que contiene las instrucciones para construir una imagen Docker. Las imágenes Docker son la base de los contenedores, y se crean a partir de los Dockerfiles mediante un proceso de construcción. Los Dockerfiles especifican qué software se incluirá en la imagen, cómo se configurará y cómo se ejecutará la aplicación.
3. **Docker Hub:** Es un registro público de imágenes Docker donde los desarrolladores pueden compartir y descargar imágenes preconstruidas para usar en sus aplicaciones. También se puede utilizar para almacenar imágenes personalizadas y privadas.

Docker Compose, por otro lado, es una herramienta que permite definir y gestionar aplicaciones multi-contenedor en Docker de manera sencilla. Con Docker Compose, puedes utilizar un archivo YAML (usualmente llamado **docker-compose.yml**) para definir la configuración de tus servicios, incluyendo la configuración de redes, volúmenes y variables de entorno. Luego, puedes utilizar el comando **docker-compose** para levantar, gestionar y detener tu aplicación de manera coordinada.

Quieres saber más de docker y sus servicios poder ir ala su página oficial <https://www.docker.com/>

Introducción

En este proyecto, vamos a desplegar una arquitectura de microservicios utilizando Docker y Docker-Compose para orquestar tres componentes esenciales que podría tener 2 o 3 servicios interconectados entre sí, en este caso será un proyecto full stack de una AgendaContactosi; el primer servicio es una base de datos PostgreSQL, una API desarrollada en Spring Boot, y una aplicación frontend desarrollada en Vue.js completamente responsive. Docker nos permite crear entornos aislados y reproducibles para cada uno de estos servicios, facilitando el desarrollo, la prueba y la implementación.

Descripción

El proyecto, denominado `my_project_docker`, consta de las siguientes carpetas y componentes:

1. `db`: contiene el archivo `backup.sql` con las instrucciones para crear las tablas e insertar datos iniciales en la base de datos PostgreSQL.
2. `demoApiAgenda`: La aplicación backend desarrollada en Spring Boot. Aquí se encuentra y código fuente de la API y el archivo `Dockerfile` para construir la imagen Docker correspondiente.
3. `app-mi-agenda`: La aplicación frontend desarrollada en Vue.js. Esta carpeta contiene el código fuente del frontend y el archivo `Dockerfile` para construir su imagen Docker.

Para comenzar a implementar Docker en este proyecto, seguiremos estos pasos:

Preparación del Entorno de trabajo:

Para verificar la instalación de Docker y Docker Compose en tu máquina, puedes seguir los pasos a continuación según el sistema operativo que estés usando: Windows 10, macOS, o Linux.

Windows 10

1. Verificar Docker:

- Abre PowerShell o el Símbolo del sistema (CMD).
- Ejecuta el siguiente comando:

```
docker --version
```

Deberías ver una salida similar a:

```
Docker version 20.10.7, build f0df350
```

2. Verificar Docker Compose:

- En la misma ventana de PowerShell o Símbolo del sistema, ejecuta:

```
docker-compose --version
```

Deberías ver una salida similar a:

```
docker-compose version 1.29.2, build 5becea4c
```

macOS

1. Verificar Docker:

- Abre la Terminal.
- Ejecuta el siguiente comando:

```
docker --version
```

Deberías ver una salida similar a:

```
Docker version 20.10.7, build f0df350
```

2. Verificar Docker Compose:

- En la misma ventana de la Terminal, ejecuta:

```
docker-compose --version
```

Deberías ver una salida similar a:

```
docker-compose version 1.29.2, build 5becea4c
```

Linux

1. Verificar Docker:

- Abre una terminal.
- Ejecuta el siguiente comando:

```
docker --version
```

Deberías ver una salida similar a:

```
Docker version 20.10.7, build f0df350
```

2. Verificar Docker Compose:

- En la misma terminal, ejecuta:

```
docker-compose --version
```

Deberías ver una salida similar a:

```
docker-compose version 1.29.2, build 5becea4c
```

Instalación del Desktop (si no está instalado)

Si no tienes Docker o Docker Compose instalados, aquí tienes una guía rápida para cada sistema operativo:

Windows 10

1. Descarga e instala Docker Desktop para Windows.
2. Durante la instalación, asegúrate de que Docker Compose esté seleccionado.
3. Sigue las instrucciones de instalación y reinicia tu máquina si es necesario.

macOS

1. Descarga e instala Docker Desktop para Mac.
2. Durante la instalación, asegúrate de que Docker Compose esté seleccionado.
3. Sigue las instrucciones de instalación.

Linux

1. Instalar Docker:

Ejecuta los siguientes comandos en la terminal:

```
sudo apt-get update
sudo apt-get install -y docker.io
sudo systemctl start docker
sudo systemctl enable docker
```

2. Instalar Docker Compose:

Ejecuta los siguientes comandos en la terminal:

```
sudo curl -L
"https://github.com/docker/compose/releases/download/1.29.
2/docker-compose-$(uname -s)-$(uname -m)" -o
/usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```


Después de seguir estos pasos, puedes verificar la instalación de Docker y Docker Compose nuevamente utilizando los comandos mencionados al principio.

Vamos a realizar la configuración con las versiones que tengo instalada en mi computadora. Para la base de datos vamos a utilizar PostgreSQL 14, para el desarrollo de la Api RESRT con Spring utilizaremos Java 17 y los puertos que especificaremos mas adelante. Realizaremos una guía detallada paso a paso para implementar Docker en nuestro proyecto:

1. Estructura del proyecto

La estructura de nuestro proyecto es la siguiente:

```
jose_@DESKTOP-KKFSQ9M MINGW64 ~/my_project_docker/
$
my_project_docker/                                #directtorio raíz de proyecto docker
|-- README.md
|-- app-mi-agenda/                                #Dockerfile para App Vue.js
|   |-- Dockerfile
|   |-- README.md
|   |-- dist/                                     #contruir dist/ para lanzarla a producción
|       |-- assets/
|       |-- index.html
|       |-- vite.svg
|   |-- index.html
|   |-- node_modules/
|   |-- package-lock.json
|   |-- package.json
|   |-- public/
|   |-- src/
|   |-- vite.config.js
|-- db/                                             ##scripts y backups sql
|   |-- backup.sql                               #Scrip creacion de tablas y insercciones
|   |-- init.sql                                 #Scrip crear Usuario y otorgar privilegios
|-- demoApiAgenda/                                #Dockerfile para Api Spring Boot
|   |-- Dockerfile
|   |-- HELP.md
|   |-- mvnw
|   |-- mvnw.cmd
|   |-- pom.xml
|   |-- src/
|   |-- target/
|       |-- demoAgenda-0.0.1-SNAPSHOT.jar
|-- docker-compose.yml                            #Docker-compose para orquestar los 3 servicios
|-- docs/
|-- ...Docker.docx                                # documentación del Reporte del proyecto(Word)

10 directories, 16 files
```

Antes de pasar a la configuración , vamos a explicar que en nuestro caso empezaremos con la configuración de nuestro archivo Docker-Compose .yaml vacío para irle agregado las configuraciones respectivas de cada servicio que utilizaremos, en este caso primero agregaremos la configuración de la base de datos en el servidor de PostgreSQL.

2. Dockerfile para la base de datos

En la carpeta **db**, no necesitamos un **Dockerfile**, pero necesitaremos un archivo de inicialización para PostgreSQL. Vamos a usar una imagen oficial de PostgreSQL en su versión 14 y un volumen para cargar el backup.

Para una configuración más segura y flexible en entorno de trabajo postgres, es recomendable usar variables de entorno para la configuración de nuestra aplicación, especialmente para datos sensibles como las credenciales de la base de datos. Además, es una buena práctica no utilizar el usuario **postgres** para aplicaciones en producción, sino crear un usuario con los privilegios necesarios.

Como en mi caso creare un usuario **'usr_admin'** con password **'usr_admin'** con sus respectivos asignación de privilegios a la base de datos **'db_agenda'**, una vez dejando esto claro avancemos con la configuración. Vamos a manejarlo de una manera que no requiera ejecutar comandos manualmente en la base de datos una vez que el contenedor esté en funcionamiento. Utilizaremos scripts de inicialización de Docker para crear el usuario y la base de datos con los privilegios necesarios.

Estos son los pasos detallados para no perdernos:

Paso 2.1. Preparar el script de inicialización para PostgreSQL

Vamos a crear un script de inicialización SQL adicional para crear el usuario `usr_admin` y otorgarle los privilegios necesarios.

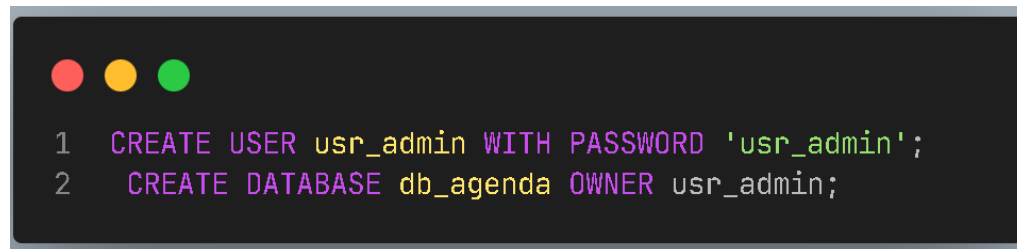
Paso 2.2: Crear el script de inicialización

En la carpeta `db` de nuestro proyecto, crea un archivo llamado `init.sql` con el siguiente contenido:

```
CREATE USER usr_admin WITH PASSWORD 'usr_admin';
```

```
CREATE DATABASE db_agenda OWNER usr_admin;
```

Debería verse así:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It displays two lines of SQL code: '1 CREATE USER usr_admin WITH PASSWORD 'usr_admin';' and '2 CREATE DATABASE db_agenda OWNER usr_admin;'.

```
1 CREATE USER usr_admin WITH PASSWORD 'usr_admin';
2 CREATE DATABASE db_agenda OWNER usr_admin;
```

Paso 2.3 Ajustar el archivo `docker-compose.yml`

Modificaremos el archivo `docker-compose.yml` para asegurarnos de que PostgreSQL ejecute este script de inicialización cuando el contenedor se levante.

Paso 2.4: Modificar `docker-compose.yml`

Añade el volumen para el nuevo script de inicialización:

```
1  version: '3.8'
2
3  services:
4    postgres_db:
5      image: postgres:14
6      container_name: postgres_db
7      environment:
8        POSTGRES_DB: db_agenda
9        POSTGRES_USER: usr_admin
10       POSTGRES_PASSWORD: usr_admin
11      volumes:
12        - ./db/backup.sql:/docker-entrypoint-initdb.d/backup.sql
13        - ./db/init.sql:/docker-entrypoint-initdb.d/init.sql
14      ports:
15        - "5432:5432"
16      #agregaremos mas configuraciones mas adelante
```

3. Configuración de la API (Spring Boot)

Paso 3.1 Creación de archivo Dockerfile

Crea un Dockerfile sin extensión en la carpeta demoApiAgenda o donde se encuentre la carpeta raíz de nuestra aplicación de Spring Boot:

Dockerfile

```
1  # Dockerfile para demoApiAgenda (Build proyectjar)
2  FROM openjdk:17-jdk-slim AS build
3
4  #crear espacio de trabajo /app
5  WORKDIR /app
6
7  #Copiar todos los archivos de configuracion y otros a la rutas raíz de nuestro proyecto
8  COPY . .
9
10 #corremos contruimos el jar de la API saltando los test de ser necesario
11 RUN ./mvnw clean package -DskipTests
12
13 #etapa de produccion (Produccion Stage)
14 FROM openjdk:17-jre-slim
15 WORKDIR /app
16
17 #copiaremos el Jar generado anteriormente a la ruta raíz de /app
18 COPY --from=build /app/target/demoAgenda-0.0.1-SNAPSHOT.jar demoApiAgenda.jar
19 #exponemos a escucha el puerto 9090 de nuestra API
20 EXPOSE 9090
21 #corremos el jar en produccion
22 ENTRYPOINT ["java", "-jar", "demoApiAgenda.jar"]
```

Paso 3.2 Crear el build para tu aplicación Spring Boot

Primero debemos modificar el archivo `'Application.properties'` de nuestro proyecto Spring Boot para establecer las variables de entorno que será establecida por nuestro `Dockfile-Compose.yml`. Por lo tanto, recordemos que, para evitar conflictos, no deberíamos usar `localhost` ni `ip` para acceder a la API desde el backend cuando se está usando conexión entre contenedores, en su lugar se usa el nombre del contenedor de la base de datos, en este caso `'postgres_db'`:

Paso 3.3 Actualización Application.properties

```
1 server.port=9090
2 #usamos variables de entorno para l conexion y sustituimos localhost o ip por el nombre del contenedor (postgres_db)
3 spring.datasource.url=${SPRING_DATASOURCE_URL:jdbc:postgresql://postgres_db:5432/db_agenda}
4 spring.datasource.username=${SPRING_DATASOURCE_USERNAME:usr_admin}
5 spring.datasource.password=${SPRING_DATASOURCE_PASSWORD:usr_admin}
6 spring.datasource.driver-class-name=org.postgresql.Driver
7 #configuraciones JPA
8 spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
9 spring.jpa.hibernate.ddl-auto=update
10 spring.jpa.show-sql=true
11 spring.jpa.properties.hibernate.format_sql=true
12 spring.application.name=api-unach-MyAgenda
13
```

Primero, compila tu aplicación Spring Boot para generar el archivo JAR que se alojará en la carpeta `target`.

1. Navega a la carpeta de tu aplicación Spring Boot desde terminal:

```
cd path/to/tu_proyectoApi
```

en mi caso:

```
cd my_project_docker/demoApiAgenda/
```


que lo compile y que lo cargue a nuestra imagen después de haberlo construido.

Paso 3.4 Actualización del archivo Docker-compose.yml

Debemos de asegurarnos que las variables de entorno estén incluidas correctamente para la API y además se pondrá un nombre al contenedor de postgres para hacer más flexible y consistente la comunicación entre contenedores y poderlos administrarlos correctamente en este caso para postgres **'postgres_db'** y para la API **'api_service'**:

Docker-compose.yml

```
1  version: '3.8'
2
3  services:
4    postgres_db:
5      image: postgres:14
6      container_name: postgres_db
7      environment:
8        POSTGRES_DB: db_agenda
9        POSTGRES_USER: usr_admin
10       POSTGRES_PASSWORD: usr_admin
11      volumes:
12        - ./db/backup.sql:/docker-entrypoint-initdb.d/backup.sql
13        - ./db/init.sql:/docker-entrypoint-initdb.d/init.sql
14      ports:
15        - "5432:5432"
16
17    api_service:
18      build:
19        context: ./demoApiAgenda
20        dockerfile: Dockerfile
21        container_name: api_service
22      environment:
23        SPRING_DATASOURCE_URL: jdbc:postgresql://postgres_db:5432/db_agenda
24        SPRING_DATASOURCE_USERNAME: usr_admin
25        SPRING_DATASOURCE_PASSWORD: usr_admin
26      depends_on:
27        - postgres_db
28      ports:
29        - "9090:9090"
```

4. Configuración de la aplicación (Vue.js)

Necesitamos construir nuestra aplicación Vue.js para generar los archivos estáticos que se alojarán en la carpeta **dist**.

1. Navega a la carpeta de tu aplicación Vue.js:
2. Instala las dependencias:
3. Crea el build de producción:

Ejecutar estos comandos:

```
1 cd app-mi-agenda
2 npm install
3 npm run build
4
```

Esto generará una carpeta **dist** con los archivos estáticos listos para producción.

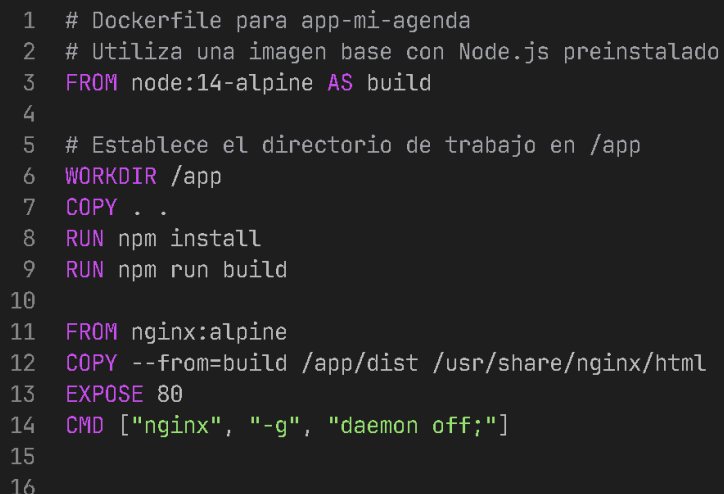
En la consola debería verse así:

```
1
2 jose_@DESKTOP-KKFSQ9M MINGW64 ~/Pictures/my_project_docker/app-mi-agenda
3 $ npm run build
4
5 > app-mi-agenda@0.0.0 build
6 > vite build
7
8 vite v5.2.11 building for production...
9 ✓ 87 modules transformed.
10 dist/index.html                                0.46 kB | gzip: 0.29 kB
11 dist/assets/fa-v4compatibility-BRdYr4HJ.woff2  4.79 kB
12 dist/assets/fa-v4compatibility-DLBX5pNp.ttf    10.83 kB
13 dist/assets/fa-regular-400-9VThgXHM.woff2      25.39 kB
14 dist/assets/fa-regular-400-C54-fRIQ.ttf        67.86 kB
15 dist/assets/fa-brands-400-Ch568Ea9.woff2       117.85 kB
16 dist/assets/fa-solid-900-QWY35r5r.woff2       156.40 kB
17 dist/assets/fa-brands-400-DHHcbFjz.ttf         209.13 kB
18 dist/assets/fa-solid-900-Cm9M9sZB.ttf          420.33 kB
19 dist/assets/index-iITdWtz_.css                 118.29 kB | gzip: 26.98 kB
20 dist/assets/index-Bc1LXBtE.js                  140.70 kB | gzip: 54.23 kB
21 ✓ built in 7.79s
22
```


Pasos finales

Paso 4.1 Creación de Dockfile:

Dockfile



```
1 # Dockerfile para app-mi-agenda
2 # Utiliza una imagen base con Node.js preinstalado
3 FROM node:14-alpine AS build
4
5 # Establece el directorio de trabajo en /app
6 WORKDIR /app
7 COPY . .
8 RUN npm install
9 RUN npm run build
10
11 FROM nginx:alpine
12 COPY --from=build /app/dist /usr/share/nginx/html
13 EXPOSE 80
14 CMD ["nginx", "-g", "daemon off;"]
15
16
```

Debemos asegurarnos que todos los servicios que realizan solicitudes HTTP al api desde nuestra aplicación frontend deben configurarse en cada servicio y este funcione sin conflictos al ejecutarse dentro de Docker, es importante no usar **localhost** para conectarse al **API**. En lugar de eso, se debe usar el nombre del servicio definido en **docker-compose.yml**, en este caso es **api_service**.

Por ejemplo, yo tengo un servicio **categorías_contactos.js** se usa el nombre del contenedor al igual que se hizo con la API **api_service** en su archivo **Application.properties**:

```

1  import axios from "axios";
2
3  const API_URL = 'http://api_service:9090/api/v1/apiMyAgenda/categorias';
4
5  export default{
6    getAllCategorias() {
7      return axios.get(API_URL)
8        .then(response => response.data)
9        .catch(error => {
10          throw error;
11        });
12    }
13  };

```

Se realizarán los cambios según sean necesarios.

Antes de subir nuestro proyecto a la imagen del contenedor necesitamos crear un archivo **‘.dockerignore’** en nuestra carpeta raíz de la aplicación **app-mi-agenda** este es opcional , dependiendo de los gustos de cada programador, pero por buena práctica lo haremos; para evitar que ciertos archivos o directorios de nuestro proyecto se incluyan en el contexto de construcción de la imagen Docker.

Ejemplo mi archivo .dockerignore de mi aplicación Vue.js:

.Dockerignore

```

1  .dockerignore
2  Dockerfile
3  node_modules
4  .vscode
5  dist
6  public

```

En este caso, estamos ignorando el Dockerfile (ya que no es necesario incluirlo en el contexto de construcción), la carpeta

node_modules (ya que Docker instalará las dependencias durante la construcción de la imagen), y las carpetas **dist** y **public** que suelen contener archivos generados que no son necesarios para la imagen Docker. Esto puede variar según la estructura y necesidad de cada proyecto que realicemos. Esto lo realice con la importancia de no incluir archivo o directorios innecesarios en la imagen Docker para mantenerla lo más liviana y eficiente posible.

Paso 4.2 Actualizar Dockfile-compose.yml

Docker-compose

```
1  version: '3.8'
2
3  services:
4    postgres_db:
5      image: postgres:14
6      container_name: postgres_db
7      environment:
8        POSTGRES_DB: db_agenda
9        POSTGRES_USER: usr_admin
10       POSTGRES_PASSWORD: usr_admin
11      volumes:
12        - ./db/backup.sql:/docker-entrypoint-initdb.d/backup.sql
13        - ./db/init.sql:/docker-entrypoint-initdb.d/init.sql
14      ports:
15        - "5432:5432"
16
17    api_service:
18      build:
19        context: ./demoApiAgenda
20        dockerfile: Dockerfile
21        container_name: api_service
22      environment:
23        SPRING_DATASOURCE_URL: jdbc:postgresql://postgres_db:5432/db_agenda
24        SPRING_DATASOURCE_USERNAME: usr_admin
25        SPRING_DATASOURCE_PASSWORD: usr_admin
26      depends_on:
27        - postgres_db
28      ports:
29        - "9090:9090"
30
31    frontend_service:
32      build:
33        context: ./app-mi-agenda
34        dockerfile: Dockerfile
35        container_name: frontend_service
36      ports:
37        - "5173:80"
38      depends_on:
39        - api_service
40
```

5.-Verificación antes de levantar el docker principal.

- 🚦 **Aplicación Spring Boot:** Verifica que el archivo JAR esté en `demoApiAgenda/target/demoApiAgenda.jar`.
- 🚦 **Aplicación Vue.js:** Verifica que los archivos de construcción estén en `App-mi-agenda/dist`.
- 🚦 **Base de datos:** Asegúrate de que `backup.sql` solo contenga la creación de tablas e inserciones, no la creación de la base de datos, también que `init.sql` tenga el script para la creación del usuario `usr_admin` y ase le asigne privilegios a la base de datos `db_agenda`.

Siguiendo estos pasos, nuestra aplicación esta lista y configurada para usar variables de entorno, lo que mejora la seguridad y flexibilidad de tu configuración. Además, con Docker Compose, todos los servicios estarán conectados entre sí y configurados para funcionar en los puertos especificados, ahora solo queda levantar el docker principal.

Como última actualización del archivo `docker-compose.yml`

Se creo la sección '`networks`',, esta configuración es crucial para que Docker pueda usar la red personalizada. Permitiendo que los servicios definidos ene l archivo se comuniquen entre sí de manera efectiva.

El archivo `'docker-compose.yml'` debería verse así:

```
1  version: '3.8'
2
3  services:
4    postgres_db:
5      image: postgres:14
6      container_name: postgres_db
7      environment:
8        POSTGRES_DB: db_agenda
9        POSTGRES_USER: usr_admin
10       POSTGRES_PASSWORD: usr_admin
11      volumes:
12        - ./db/backup.sql:/docker-entrypoint-initdb.d/backup.sql
13        - ./db/init.sql:/docker-entrypoint-initdb.d/init.sql
14      ports:
15        - "5432:5432"
16      networks:
17        - my_network
18
19    api_service:
20      build:
21        context: ./demoApiAgenda
22        dockerfile: Dockerfile
23      container_name: api_service
24      environment:
25        SPRING_DATASOURCE_URL: jdbc:postgresql://postgres_db:5432/db_agenda
26        SPRING_DATASOURCE_USERNAME: usr_admin
27        SPRING_DATASOURCE_PASSWORD: usr_admin
28      depends_on:
29        - postgres_db
30      ports:
31        - "9090:9090"
32      networks:
33        - my_network
34
35    frontend_service:
36      build:
37        context: ./app-mi-agenda
38        dockerfile: Dockerfile
39      container_name: frontend_service
40      ports:
41        - "5173:80"
42      depends_on:
43        - api_service
44      networks:
45        - my_network
46
47    networks:
48      my_network:
49        driver: bridge
50
```

Vamos a explicar la sección `'networks'`:

1.-Definicion de Red:

```
46
47 networks:
48   my_network:
49     driver: bridge
50
```

- ✚ **'networks'**: Esta clave define todas las redes que desees utilizar en tu configuración de docker-compose.
- ✚ **'my_network'**: este es el nombre de la red que se utilizara. Podemos configurarla por un nombre que deseemos , en este caso elegí uno más sencillo.
- ✚ **'driver: bridge'**: El controlador **'bridge'**

Crea una red de puente estándar que permite que los contenedores se comuniquen entre sí en la misma red. Este es el tipo de red más común para configuraciones de red simple.

2.-Asignacion de Red a Servicios:

Cada servicio tiene la clave **'networks'** donde se especifica a que red debe conectarse.

```
1  services:
2    postgres_db:
3      ...
4      networks:
5        - my_network
6
7    api_service:
8      ...
9      networks:
10       - my_network
11
12    frontend_service:
13      ...
14      networks:
15        - my_network
16
```

Esto indica que **'postgres_db'**, **'api_service'**, **'frontend service'** se conectara a la red **'my_network'**. Esto es crucial para que los contenedores puedan comunicarse entre sí utilizando los nombres de servicio definidos.

Comunicación entre Contenedores

Al usar la red `my_network`, los servicios pueden referirse entre sí por sus nombres de servicio definidos en el archivo `docker-compose.yml`. Por ejemplo:

- El contenedor `api_service` puede conectarse a la base de datos PostgreSQL usando `postgres_db` como el hostname.
- La aplicación frontend puede comunicarse con la API backend usando `api_service` como el hostname.

6.- Iniciar los contenedores con Docker Compose: Desde la raíz del proyecto:

`docker-compose up --build`

Debería salir algo como esto:

```
1 # Docker Compose file
2
3 # Name of the application
4 name: my_app
5
6 # Services
7 services:
8   # Frontend service
9   frontend:
10     build: .
11     ports:
12       - 3000:3000
13     networks:
14       - my_network
15
16   # Backend service
17   api_service:
18     build: .
19     ports:
20       - 5000:5000
21     networks:
22       - my_network
23
24   # Database service
25   postgres_db:
26     image: postgres:13
27     ports:
28       - 5432:5432
29     environment:
30       POSTGRES_DB: my_app
31     networks:
32       - my_network
33
34   # Redis service
35   redis:
36     image: redis:7
37     ports:
38       - 6379:6379
39     networks:
40       - my_network
41
42   # Elasticsearch service
43   elasticsearch:
44     image: elasticsearch:7.10.2
45     ports:
46       - 9200:9200
47       - 9300:9300
48     environment:
49       ES_JAVA_OPTS: -Xms512m -Xmx512m
50     networks:
51       - my_network
52
53   # Kibana service
54   kibana:
55     image: kibana:7.10.2
56     ports:
57       - 5601:5601
58     networks:
59       - my_network
60
61   # Logstash service
62   logstash:
63     image: logstash:7.10.2
64     ports:
65       - 5044:5044
66     networks:
67       - my_network
68
69   # Solr service
70   solr:
71     image: solr:8.11.0
72     ports:
73       - 8983:8983
74     networks:
75       - my_network
76
77   # Zookeeper service
78   zookeeper:
79     image: zookeeper:3.6.0
80     ports:
81       - 2181:2181
82     networks:
83       - my_network
84
85   # Hadoop service
86   hadoop:
87     image: hadoop:3.2.4
88     ports:
89       - 8020:8020
90     networks:
91       - my_network
92
93   # Hive service
94   hive:
95     image: hive:3.12.0
96     ports:
97       - 9083:9083
98     networks:
99       - my_network
100
101   # Pig service
102   pig:
103     image: pig:0.16.0
104     ports:
105       - 5050:5050
106     networks:
107       - my_network
108
109   # Tez service
110   tez:
111     image: tez:0.8.4
112     ports:
113       - 8030:8030
114     networks:
115       - my_network
116
117   # Mahout service
118   mahout:
119     image: mahout:0.12.2
120     ports:
121       - 5454:5454
122     networks:
123       - my_network
124
125   # HBase service
126   hbase:
127     image: hbase:2.2.1
128     ports:
129       - 16010:16010
130     networks:
131       - my_network
132
133   # Spark service
134   spark:
135     image: spark:3.0.0
136     ports:
137       - 7077:7077
138     networks:
139       - my_network
140
141   # Flink service
142   flink:
143     image: flink:1.10.0
144     ports:
145       - 6123:6123
146     networks:
147       - my_network
148
149   # Storm service
150   storm:
151     image: storm:2.4.0
152     ports:
153       - 6724:6724
154     networks:
155       - my_network
156
157   # Mesos service
158   mesos:
159     image: mesos:0.25.0
160     ports:
161       - 5050:5050
162     networks:
163       - my_network
164
165   # Kubernetes service
166   kubernetes:
167     image: kubernetes:1.19.0
168     ports:
169       - 8080:8080
170     networks:
171       - my_network
172
173   # Docker service
174   docker:
175     image: docker:20.10.12
176     ports:
177       - 2376:2376
178     networks:
179       - my_network
180
181   # Jenkins service
182   jenkins:
183     image: jenkins:2.345.3
184     ports:
185       - 8080:8080
186     networks:
187       - my_network
188
189   # GitLab service
190   gitlab:
191     image: gitlab:14.0.0
192     ports:
193       - 8080:8080
194     networks:
195       - my_network
196
197   # Next.js service
198   nextjs:
199     image: nextjs:12.0.1
200     ports:
201       - 3000:3000
202     networks:
203       - my_network
204
205   # React service
206   react:
207     image: react:18.0.0
208     ports:
209       - 3000:3000
210     networks:
211       - my_network
212
213   # Vue service
214   vue:
215     image: vue:3.0.0
216     ports:
217       - 3000:3000
218     networks:
219       - my_network
220
221   # Angular service
222   angular:
223     image: angular:12.0.0
224     ports:
225       - 3000:3000
226     networks:
227       - my_network
228
229   # Svelte service
230   svelte:
231     image: svelte:3.0.0
232     ports:
233       - 3000:3000
234     networks:
235       - my_network
236
237   # SolidJS service
238   solidjs:
239     image: solidjs:1.0.0
240     ports:
241       - 3000:3000
242     networks:
243       - my_network
244
245   # SvelteKit service
246   sveltekit:
247     image: sveltekit:1.0.0
248     ports:
249       - 3000:3000
250     networks:
251       - my_network
252
253   # Nuxt service
254   nuxt:
255     image: nuxt:3.0.0
256     ports:
257       - 3000:3000
258     networks:
259       - my_network
260
261   # Gatsby service
262   gatsby:
263     image: gatsby:3.0.0
264     ports:
265       - 3000:3000
266     networks:
267       - my_network
268
269   # Next.js service
270   nextjs:
271     image: nextjs:12.0.1
272     ports:
273       - 3000:3000
274     networks:
275       - my_network
276
277   # React service
278   react:
279     image: react:18.0.0
280     ports:
281       - 3000:3000
282     networks:
283       - my_network
284
285   # Vue service
286   vue:
287     image: vue:3.0.0
288     ports:
289       - 3000:3000
290     networks:
291       - my_network
292
293   # Angular service
294   angular:
295     image: angular:12.0.0
296     ports:
297       - 3000:3000
298     networks:
299       - my_network
300
301   # Svelte service
302   svelte:
303     image: svelte:3.0.0
304     ports:
305       - 3000:3000
306     networks:
307       - my_network
308
309   # SolidJS service
310   solidjs:
311     image: solidjs:1.0.0
312     ports:
313       - 3000:3000
314     networks:
315       - my_network
316
317   # SvelteKit service
318   sveltekit:
319     image: sveltekit:1.0.0
320     ports:
321       - 3000:3000
322     networks:
323       - my_network
324
325   # Nuxt service
326   nuxt:
327     image: nuxt:3.0.0
328     ports:
329       - 3000:3000
330     networks:
331       - my_network
332
333   # Gatsby service
334   gatsby:
335     image: gatsby:3.0.0
336     ports:
337       - 3000:3000
338     networks:
339       - my_network
340
341   # Next.js service
342   nextjs:
343     image: nextjs:12.0.1
344     ports:
345       - 3000:3000
346     networks:
347       - my_network
348
349   # React service
350   react:
351     image: react:18.0.0
352     ports:
353       - 3000:3000
354     networks:
355       - my_network
356
357   # Vue service
358   vue:
359     image: vue:3.0.0
360     ports:
361       - 3000:3000
362     networks:
363       - my_network
364
365   # Angular service
366   angular:
367     image: angular:12.0.0
368     ports:
369       - 3000:3000
370     networks:
371       - my_network
372
373   # Svelte service
374   svelte:
375     image: svelte:3.0.0
376     ports:
377       - 3000:3000
378     networks:
379       - my_network
380
381   # SolidJS service
382   solidjs:
383     image: solidjs:1.0.0
384     ports:
385       - 3000:3000
386     networks:
387       - my_network
388
389   # SvelteKit service
390   sveltekit:
391     image: sveltekit:1.0.0
392     ports:
393       - 3000:3000
394     networks:
395       - my_network
396
397   # Nuxt service
398   nuxt:
399     image: nuxt:3.0.0
400     ports:
401       - 3000:3000
402     networks:
403       - my_network
404
405   # Gatsby service
406   gatsby:
407     image: gatsby:3.0.0
408     ports:
409       - 3000:3000
410     networks:
411       - my_network
412
413   # Next.js service
414   nextjs:
415     image: nextjs:12.0.1
416     ports:
417       - 3000:3000
418     networks:
419       - my_network
420
421   # React service
422   react:
423     image: react:18.0.0
424     ports:
425       - 3000:3000
426     networks:
427       - my_network
428
429   # Vue service
430   vue:
431     image: vue:3.0.0
432     ports:
433       - 3000:3000
434     networks:
435       - my_network
436
437   # Angular service
438   angular:
439     image: angular:12.0.0
440     ports:
441       - 3000:3000
442     networks:
443       - my_network
444
445   # Svelte service
446   svelte:
447     image: svelte:3.0.0
448     ports:
449       - 3000:3000
450     networks:
451       - my_network
452
453   # SolidJS service
454   solidjs:
455     image: solidjs:1.0.0
456     ports:
457       - 3000:3000
458     networks:
459       - my_network
460
461   # SvelteKit service
462   sveltekit:
463     image: sveltekit:1.0.0
464     ports:
465       - 3000:3000
466     networks:
467       - my_network
468
469   # Nuxt service
470   nuxt:
471     image: nuxt:3.0.0
472     ports:
473       - 3000:3000
474     networks:
475       - my_network
476
477   # Gatsby service
478   gatsby:
479     image: gatsby:3.0.0
480     ports:
481       - 3000:3000
482     networks:
483       - my_network
484
485   # Next.js service
486   nextjs:
487     image: nextjs:12.0.1
488     ports:
489       - 3000:3000
490     networks:
491       - my_network
492
493   # React service
494   react:
495     image: react:18.0.0
496     ports:
497       - 3000:3000
498     networks:
499       - my_network
500
501   # Vue service
502   vue:
503     image: vue:3.0.0
504     ports:
505       - 3000:3000
506     networks:
507       - my_network
508
509   # Angular service
510   angular:
511     image: angular:12.0.0
512     ports:
513       - 3000:3000
514     networks:
515       - my_network
516
517   # Svelte service
518   svelte:
519     image: svelte:3.0.0
520     ports:
521       - 3000:3000
522     networks:
523       - my_network
524
525   # SolidJS service
526   solidjs:
527     image: solidjs:1.0.0
528     ports:
529       - 3000:3000
530     networks:
531       - my_network
532
533   # SvelteKit service
534   sveltekit:
535     image: sveltekit:1.0.0
536     ports:
537       - 3000:3000
538     networks:
539       - my_network
540
541   # Nuxt service
542   nuxt:
543     image: nuxt:3.0.0
544     ports:
545       - 3000:3000
546     networks:
547       - my_network
548
549   # Gatsby service
550   gatsby:
551     image: gatsby:3.0.0
552     ports:
553       - 3000:3000
554     networks:
555       - my_network
556
557   # Next.js service
558   nextjs:
559     image: nextjs:12.0.1
560     ports:
561       - 3000:3000
562     networks:
563       - my_network
564
565   # React service
566   react:
567     image: react:18.0.0
568     ports:
569       - 3000:3000
570     networks:
571       - my_network
572
573   # Vue service
574   vue:
575     image: vue:3.0.0
576     ports:
577       - 3000:3000
578     networks:
579       - my_network
580
581   # Angular service
582   angular:
583     image: angular:12.0.0
584     ports:
585       - 3000:3000
586     networks:
587       - my_network
588
589   # Svelte service
590   svelte:
591     image: svelte:3.0.0
592     ports:
593       - 3000:3000
594     networks:
595       - my_network
596
597   # SolidJS service
598   solidjs:
599     image: solidjs:1.0.0
600     ports:
601       - 3000:3000
602     networks:
603       - my_network
604
605   # SvelteKit service
606   sveltekit:
607     image: sveltekit:1.0.0
608     ports:
609       - 3000:3000
610     networks:
611       - my_network
612
613   # Nuxt service
614   nuxt:
615     image: nuxt:3.0.0
616     ports:
617       - 3000:3000
618     networks:
619       - my_network
620
621   # Gatsby service
622   gatsby:
623     image: gatsby:3.0.0
624     ports:
625       - 3000:3000
626     networks:
627       - my_network
628
629   # Next.js service
630   nextjs:
631     image: nextjs:12.0.1
632     ports:
633       - 3000:3000
634     networks:
635       - my_network
636
637   # React service
638   react:
639     image: react:18.0.0
640     ports:
641       - 3000:3000
642     networks:
643       - my_network
644
645   # Vue service
646   vue:
647     image: vue:3.0.0
648     ports:
649       - 3000:3000
650     networks:
651       - my_network
652
653   # Angular service
654   angular:
655     image: angular:12.0.0
656     ports:
657       - 3000:3000
658     networks:
659       - my_network
660
661   # Svelte service
662   svelte:
663     image: svelte:3.0.0
664     ports:
665       - 3000:3000
666     networks:
667       - my_network
668
669   # SolidJS service
670   solidjs:
671     image: solidjs:1.0.0
672     ports:
673       - 3000:3000
674     networks:
675       - my_network
676
677   # SvelteKit service
678   sveltekit:
679     image: sveltekit:1.0.0
680     ports:
681       - 3000:3000
682     networks:
683       - my_network
684
685   # Nuxt service
686   nuxt:
687     image: nuxt:3.0.0
688     ports:
689       - 3000:3000
690     networks:
691       - my_network
692
693   # Gatsby service
694   gatsby:
695     image: gatsby:3.0.0
696     ports:
697       - 3000:3000
698     networks:
699       - my_network
700
701   # Next.js service
702   nextjs:
703     image: nextjs:12.0.1
704     ports:
705       - 3000:3000
706     networks:
707       - my_network
708
709   # React service
710   react:
711     image: react:18.0.0
712     ports:
713       - 3000:3000
714     networks:
715       - my_network
716
717   # Vue service
718   vue:
719     image: vue:3.0.0
720     ports:
721       - 3000:3000
722     networks:
723       - my_network
724
725   # Angular service
726   angular:
727     image: angular:12.0.0
728     ports:
729       - 3000:3000
730     networks:
731       - my_network
732
733   # Svelte service
734   svelte:
735     image: svelte:3.0.0
736     ports:
737       - 3000:3000
738     networks:
739       - my_network
740
741   # SolidJS service
742   solidjs:
743     image: solidjs:1.0.0
744     ports:
745       - 3000:3000
746     networks:
747       - my_network
748
749   # SvelteKit service
750   sveltekit:
751     image: sveltekit:1.0.0
752     ports:
753       - 3000:3000
754     networks:
755       - my_network
756
757   # Nuxt service
758   nuxt:
759     image: nuxt:3.0.0
760     ports:
761       - 3000:3000
762     networks:
763       - my_network
764
765   # Gatsby service
766   gatsby:
767     image: gatsby:3.0.0
768     ports:
769       - 3000:3000
770     networks:
771       - my_network
772
773   # Next.js service
774   nextjs:
775     image: nextjs:12.0.1
776     ports:
777       - 3000:3000
778     networks:
779       - my_network
780
781   # React service
782   react:
783     image: react:18.0.0
784     ports:
785       - 3000:3000
786     networks:
787       - my_network
788
789   # Vue service
790   vue:
791     image: vue:3.0.0
792     ports:
793       - 3000:3000
794     networks:
795       - my_network
796
797   # Angular service
798   angular:
799     image: angular:12.0.0
800     ports:
801       - 3000:3000
802     networks:
803       - my_network
804
805   # Svelte service
806   svelte:
807     image: svelte:3.0.0
808     ports:
809       - 3000:3000
810     networks:
811       - my_network
812
813   # SolidJS service
814   solidjs:
815     image: solidjs:1.0.0
816     ports:
817       - 3000:3000
818     networks:
819       - my_network
820
821   # SvelteKit service
822   sveltekit:
823     image: sveltekit:1.0.0
824     ports:
825       - 3000:3000
826     networks:
827       - my_network
828
829   # Nuxt service
830   nuxt:
831     image: nuxt:3.0.0
832     ports:
833       - 3000:3000
834     networks:
835       - my_network
836
837   # Gatsby service
838   gatsby:
839     image: gatsby:3.0.0
840     ports:
841       - 3000:3000
842     networks:
843       - my_network
844
845   # Next.js service
846   nextjs:
847     image: nextjs:12.0.1
848     ports:
849       - 3000:3000
850     networks:
851       - my_network
852
853   # React service
854   react:
855     image: react:18.0.0
856     ports:
857       - 3000:3000
858     networks:
859       - my_network
860
861   # Vue service
862   vue:
863     image: vue:3.0.0
864     ports:
865       - 3000:3000
866     networks:
867       - my_network
868
869   # Angular service
870   angular:
871     image: angular:12.0.0
872     ports:
873       - 3000:3000
874     networks:
875       - my_network
876
877   # Svelte service
878   svelte:
879     image: svelte:3.0.0
880     ports:
881       - 3000:3000
882     networks:
883       - my_network
884
885   # SolidJS service
886   solidjs:
887     image: solidjs:1.0.0
888     ports:
889       - 3000:3000
890     networks:
891       - my_network
892
893   # SvelteKit service
894   sveltekit:
895     image: sveltekit:1.0.0
896     ports:
897       - 3000:3000
898     networks:
899       - my_network
900
901   # Nuxt service
902   nuxt:
903     image: nuxt:3.0.0
904     ports:
905       - 3000:3000
906     networks:
907       - my_network
908
909   # Gatsby service
910   gatsby:
911     image: gatsby:3.0.0
912     ports:
913       - 3000:3000
914     networks:
915       - my_network
916
917   # Next.js service
918   nextjs:
919     image: nextjs:12.0.1
920     ports:
921       - 3000:3000
922     networks:
923       - my_network
924
925   # React service
926   react:
927     image: react:18.0.0
928     ports:
929       - 3000:3000
930     networks:
931       - my_network
932
933   # Vue service
934   vue:
935     image: vue:3.0.0
936     ports:
937       - 3000:3000
938     networks:
939       - my_network
940
941   # Angular service
942   angular:
943     image: angular:12.0.0
944     ports:
945       - 3000:3000
946     networks:
947       - my_network
948
949   # Svelte service
950   svelte:
951     image: svelte:3.0.0
952     ports:
953       - 3000:3000
954     networks:
955       - my_network
956
957   # SolidJS service
958   solidjs:
959     image: solidjs:1.0.0
960     ports:
961       - 3000:3000
962     networks:
963       - my_network
964
965   # SvelteKit service
966   sveltekit:
967     image: sveltekit:1.0.0
968     ports:
969       - 3000:3000
970     networks:
971       - my_network
972
973   # Nuxt service
974   nuxt:
975     image: nuxt:3.0.0
976     ports:
977       - 3000:3000
978     networks:
979       - my_network
980
981   # Gatsby service
982   gatsby:
983     image: gatsby:3.0.0
984     ports:
985       - 3000:3000
986     networks:
987       - my_network
988
989   # Next.js service
990   nextjs:
991     image: nextjs:12.0.1
992     ports:
993       - 3000:3000
994     networks:
995       - my_network
996
997   # React service
998   react:
999     image: react:18.0.0
1000     ports:
1001       - 3000:3000
1002     networks:
1003       - my_network
1004
1005   # Vue service
1006   vue:
1007     image: vue:3.0.0
1008     ports:
1009       - 3000:3000
1010     networks:
1011       - my_network
1012
1013   # Angular service
1014   angular:
1015     image: angular:12.0.0
1016     ports:
1017       - 3000:3000
1018     networks:
1019       - my_network
1020
1021   # Svelte service
1022   svelte:
1023     image: svelte:3.0.0
1024     ports:
1025       - 3000:3000
1026     networks:
1027       - my_network
1028
1029   # SolidJS service
1030   solidjs:
1031     image: solidjs:1.0.0
1032     ports:
1033       - 3000:3000
1034     networks:
1035       - my_network
1036
1037   # SvelteKit service
1038   sveltekit:
1039     image: sveltekit:1.0.0
1040     ports:
1041       - 3000:3000
1042     networks:
1043       - my_network
1044
1045   # Nuxt service
1046   nuxt:
1047     image: nuxt:3.0.0
1048     ports:
1049       - 3000:3000
1050     networks:
1051       - my_network
1052
1053   # Gatsby service
1054   gatsby:
1055     image: gatsby:3.0.0
1056     ports:
1057       - 3000:3000
1058     networks:
1059       - my_network
1060
1061   # Next.js service
1062   nextjs:
1063     image: nextjs:12.0.1
1064     ports:
1065       - 3000:3000
1066     networks:
1067       - my_network
1068
1069   # React service
1070   react:
1071     image: react:18.0.0
1072     ports:
1073       - 3000:3000
1074     networks:
1075       - my_network
1076
1077   # Vue service
1078   vue:
1079     image: vue:3.0.0
1080     ports:
1081       - 3000:3000
1082     networks:
1083       - my_network
1084
1085   # Angular service
1086   angular:
1087     image: angular:12.0.0
1088     ports:
1089       - 3000:3000
1090     networks:
1091       - my_network
1092
1093   # Svelte service
1094   svelte:
1095     image: svelte:3.0.0
1096     ports:
1097       - 3000:3000
1098     networks:
1099       - my_network
1100
1101   # SolidJS service
1102   solidjs:
1103     image: solidjs:1.0.0
1104     ports:
1105       - 3000:3000
1106     networks:
1107       - my_network
1108
1109   # SvelteKit service
1110   sveltekit:
1111     image: sveltekit:1.0.0
1112     ports:
1113       - 3000:3000
1114     networks:
1115       - my_network
1116
1117   # Nuxt service
1118   nuxt:
1119     image: nuxt:3.0.0
1120     ports:
1121       - 3000:3000
1122     networks:
1123       - my_network
1124
1125   # Gatsby service
1126   gatsby:
1127     image: gatsby:3.0.0
1128     ports:
1129       - 3000:3000
1130     networks:
1131       - my_network
1132
1133   # Next.js service
1134   nextjs:
1135     image: nextjs:12.0.1
1136     ports:
1137       - 3000:3000
1138     networks:
1139       - my_network
1140
1141   # React service
1142   react:
1143     image: react:18.0.0
1144     ports:
1145       - 3000:3000
1146     networks:
1147       - my_network
1148
1149   # Vue service
1150   vue:
1151     image: vue:3.0.0
1152     ports:
1153       - 3000:3000
1154     networks:
1155       - my_network
1156
1157   # Angular service
1158   angular:
1159     image: angular:12.0.0
1160     ports:
1161       - 3000:3000
1162     networks:
1163       - my_network
1164
1165   # Svelte service
1166   svelte:
1167     image: svelte:3.0.0
1168     ports:
1169       - 3000:3000
1170     networks:
1171       - my_network
1172
1173   # SolidJS service
1174   solidjs:
1175     image: solidjs:1.0.0
1176     ports:
1177       - 3000:3000
1178     networks:
1179       - my_network
1180
1181   # SvelteKit service
1182   sveltekit:
1183     image: sveltekit:1.0.0
1184     ports:
1185       - 3000:3000
1186     networks:
1187       - my_network
1188
1189   # Nuxt service
1190   nuxt:
1191     image: nuxt:3.0.0
1192     ports:
1193       - 3000:3000
1194     networks:
1195       - my_network
1196
1197   # Gatsby service
1198   gatsby:
1199     image: gatsby:3.0.0
1200     ports:
1201       - 3000:3000
1202     networks:
1203       - my_network
1204
1205   # Next.js service
1206   nextjs:
1207     image: nextjs:12.0.1
1208     ports:
1209       - 3000:3000
1210     networks:
1211       - my_network
1212
1213   # React service
1214   react:
1215     image: react:18.0.0
1216     ports:
1217       - 3000:3000
1218     networks:
1219       - my_network
1220
1221   # Vue service
1222   vue:
1223     image: vue:3.0.0
1224     ports:
1225       - 3000:3000
1226     networks:
1227       - my_network
1228
1229   # Angular service
1230   angular:
1231     image: angular:12.0.0
1232     ports:
1233       - 3000:3000
1234     networks:
1235       - my_network
1236
1237   # Svelte service
1238   svelte:
1239     image: svelte:3.0.0
1240     ports:
1241       - 3000:3000
1242     networks:
1243       - my_network
1244
1245   # SolidJS service
1246   solidjs:
1247     image: solidjs:1.0.0
1248     ports:
1249       - 3000:3000
1250     networks:
1251       - my_network
1252
1253   # SvelteKit service
1254   sveltekit:
1255     image: sveltekit:1.0.0
1256     ports:
1257       - 3000:3000
1258     networks:
1259       - my_network
1260
1261   # Nuxt service
1262   nuxt:
1263     image: nuxt:3.0.0
1264     ports:
1265       - 3000:3000
1266     networks:
1267       - my_network
1268
1269   # Gatsby service
1270   gatsby:
1271     image: gatsby:3.0.0
1272     ports:
1273       - 3000:3000
1274     networks:
1275       - my_network
1276
1277   # Next.js service
1278   nextjs:
1279     image: nextjs:12.0.1
1280     ports:
1281       - 3000:3000
1282     networks:
1283       - my_network
1284
1285   # React service
1286   react:
128
```

Estructura Final

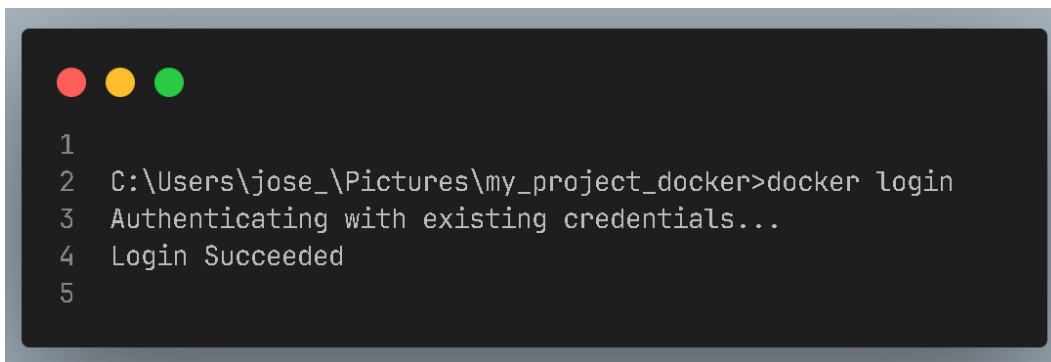
1.- Estructura del proyecto:

```
my_project_docker/
├─ README.md
├─ app-mi-agenda
│   ├─ Dockerfile
│   ├─ README.md
│   ├─ dist
│   ├─ index.html
│   ├─ node_modules
│   ├─ package-lock.json
│   ├─ package.json
│   ├─ public
│   ├─ src
│   └─ vite.config.js
├─ db
│   ├─ backup.sql
│   └─ init.sql
├─ demoApiAgenda
│   ├─ Dockerfile
│   ├─ HELP.md
│   ├─ mvnw
│   ├─ mvnw.cmd
│   ├─ pom.xml
│   ├─ src
│   └─ target
├─ docker-compose.yml
└─ docs
    ├─ Instalacion-Docker-&-Implementacion-ServiceFullStack-Agenda-With-Docker.docx
    └─ snap_code
```


Subir y Compartir Los repositorios de Docker Hub

Para compartir tus imágenes de Docker con alguna persona para que pueda calificar y probar los contenedores en su máquina, puedes seguir estos pasos:

Antes de los comenzar debemos logearnos desde nuestra terminal, Ejemplo: En mi caso tengo instalado Docker Desktop e iniciado sesión. Por ende, no me salteo la confirmación de credenciales. En tu caso de no estar logeado en la aplicación Desktop te solicitara las credenciales.

A screenshot of a Windows command prompt terminal window. The window has a dark background and a light gray title bar. In the top left corner, there are three colored window control buttons: red, yellow, and green. The terminal shows a series of five numbered lines. Line 1 is a blank line. Line 2 shows the command 'C:\Users\jose_\Pictures\my_project_docker>docker login'. Line 3 shows the output 'Authenticating with existing credentials...'. Line 4 shows the output 'Login Succeeded'. Line 5 is a blank line.

```
1
2 C:\Users\jose_\Pictures\my_project_docker>docker login
3 Authenticating with existing credentials...
4 Login Succeeded
5
```

Paso 1: Subir las Imágenes a Docker Hub

Como ya hemos discutido, primero necesitas subir tus imágenes a Docker Hub. Aquí está un resumen rápido de los pasos para hacerlo:

1. Construir las imágenes (si aún no lo has hecho):

`docker-compose build`

```
1 C:\Users\jose_\Pictures\my_project_docker>docker compose build
2 time="2024-05-19T21:17:38-06:00" level=warning msg="C:\Users\jose_\Pictures\my_project_docker\docker-compose.yml: 'version' is obsolete"
3 [+] Building 0.0s (0/0)  docker:default
4 [+] Building 7.3s (24/24) FINISHED
5 => [api_service internal] load build definition from Dockerfile
6 => => transferring dockerfile: 778B
7 => [api_service internal] load metadata for docker.io/library/openjdk:17-jdk-alpine
8 => [api_service auth] library/openjdk:pull token for registry-1.docker.io
9 => [api_service internal] load .dockerignore
10 => => transferring context: 28B
11 => [api_service internal] load build context
12 => => transferring context: 28.0kB
13 => [api_service build 1/4] FROM docker.io/library/openjdk:17-jdk-alpine@sha256:4b6abae565492dbe9e7a894137c966a7485154238902f2f25e9dbd9784385d81
14 => CACHED [api_service build 2/4] WORKDIR /app
15 => CACHED [api_service build 3/4] COPY . .
16 => CACHED [api_service build 4/4] RUN ./mvnw clean package -DskipTests
17 => CACHED [api_service stage-1 3/3] COPY --from=build /app/target/demoAgenda-0.0.1-SNAPSHOT.jar demoApiAgenda.jar
18 => [api_service] exporting to image
19 => => exporting layers
20 => => writing image sha256:4fe17a7340894bbef1603eacceaed35741c83655805c191654439a494c31cd5b
21 => => naming to docker.io/library/my_project_docker-api_service
22 => [frontend_service internal] load build definition from Dockerfile
23 => => transferring dockerfile: 335B
24 => [frontend_service internal] load metadata for docker.io/library/node:20-alpine
25 => [frontend_service auth] library/node:pull token for registry-1.docker.io
26 => [frontend_service internal] load .dockerignore
27 => => transferring context: 102B
28 => [frontend_service internal] load build context
29 => => transferring context: 1.08kB
30 => [frontend_service build 1/5] FROM docker.io/library/node:20-alpine@sha256:291e84d956f1aff38454bbd3da38941461ad569a185c20aa289f71f37ea08e23
31 => CACHED [frontend_service build 2/5] WORKDIR /app
32 => CACHED [frontend_service build 3/5] COPY . .
33 => CACHED [frontend_service build 4/5] RUN npm install
34 => CACHED [frontend_service build 5/5] RUN npm run build
35 => CACHED [frontend_service stage-1 3/4] COPY --from=build /app/dist ./dist
36 => CACHED [frontend_service stage-1 4/4] RUN npm install -g http-server
37 => [frontend_service] exporting to image
38 => => exporting layers
39 => => writing image sha256:3e0259858c7aee9677717962018f6e62029e7d8deb08ce3da50d57a2d3b0d1eb
40 => => naming to docker.io/library/my_project_docker-frontend_service
41
```

verificamos que las imágenes se hayan completado exitosamente:

```
1 C:\Users\jose_\Pictures\my_project_docker>docker images
2 REPOSITORY          TAG         IMAGE ID      CREATED       SIZE
3 my_project_docker-api_service   latest     4fe17a734089  21 hours ago  371MB
4 my_project_docker-frontend_service latest     3e0259858c7a  21 hours ago  143MB
5 postgres             14         b7c9cffffa669 10 days ago   422MB
```

2. Taguear(asignar tag) las imágenes para Docker Hub:

```
docker tag my_project_docker-frontend_service:latest
1603pepe/my_project_docker-frontend_service:1.0

docker tag my_project_docker-api_service:latest
1603pepe/my_project_docker-api_service:1.0

docker tag postgres_db:14 1603pepe/my_project_docker-
postgres_db:1.0
```

verificamos nuevamente:

```
1
2
3 C:\Users\jose\Pictures\my_project_docker>docker tag my_project_docker-frontend_service:latest 1603pepe/my_project_docker-frontend_service:1.0
4
5 C:\Users\jose\Pictures\my_project_docker>docker tag my_project_docker-api_service:latest 1603pepe/my_project_docker-api_service:1.0
6
7 C:\Users\jose\Pictures\my_project_docker>docker tag my_project_docker-postgres_db:latest 1603pepe/my_project_docker-postgres_db:1.0
8 Error response from daemon: No such image: my_project_docker-postgres_db:latest
9
10 C:\Users\jose\Pictures\my_project_docker>docker tag my_project_docker-frontend_service:latest 1603pepe/my_project_docker-frontend_service:1.0
11
12 C:\Users\jose\Pictures\my_project_docker>docker tag my_project_docker-api_service:latest 1603pepe/my_project_docker-api_service:1.0
13
14 C:\Users\jose\Pictures\my_project_docker>docker tag my_project_docker-postgres_db:latest 1603pepe/my_project_docker-postgres_db:1.0
15 Error response from daemon: No such image: my_project_docker-postgres_db:latest
16
17 C:\Users\jose\Pictures\my_project_docker>docker tag postgres 1603pepe/my_project_docker-postgres_db:1.0
18 Error response from daemon: No such image: postgres:latest
19
20 C:\Users\jose\Pictures\my_project_docker>docker tag postgres:14 1603pepe/my_project_docker-postgres_db:1.0
21
22 C:\Users\jose\Pictures\my_project_docker>docker images
23 REPOSITORY          TAG         IMAGE ID      CREATED       SIZE
24 1603pepe/my_project_docker-api_service    1.0         4fe17a734089  21 hours ago  371MB
25 my_project_docker-api_service              latest      4fe17a734089  21 hours ago  371MB
26 1603pepe/my_project_docker-frontend_service 1.0         3e0259858c7a  21 hours ago  143MB
27 my_project_docker-frontend_service          latest      3e0259858c7a  21 hours ago  143MB
28 1603pepe/my_project_docker-postgres_db      1.0         b7cbcfffa669  10 days ago   422MB
29 postgres                                   14         b7cbcfffa669  10 days ago   422MB
```

3. Subir las imágenes a Docker Hub:

`docker push your_dockerhub_username/my_project_docker-images:versión`

en la terminal debería de aparecer algo como esto:

```
1                                     14          b7cbcfffa669   10 days ago   422MB
2
3 C:\Users\jose\Pictures\my_project_docker>docker push 1603pepe/my_project_docker-postgres_db:1.0
4 The push refers to repository [docker.io/1603pepe/my_project_docker-postgres_db]
5 19209b659da8: Mounted from library/postgres
6 5f28efc87628: Mounted from library/postgres
7 14ba3ffbc633: Mounted from library/postgres
8 cbce9a667cfe: Mounted from library/postgres
9 a1eccf2329b0: Mounted from library/postgres
10 3c94ec32df08: Pushed
11 eb729ffb640a: Mounted from library/postgres
12 ca0d20cf05c2: Mounted from library/postgres
13 6107b7a8ff44: Mounted from library/postgres
14 31b9a2ba6622: Mounted from library/postgres
15 4305ace4d869: Mounted from library/postgres
16 f19a0a4c2bad: Mounted from library/postgres
17 77d3b9ed4c37: Mounted from library/postgres
18 5d4427064ecc: Mounted from library/postgres
19 1.0: digest: sha256:a107b6c592fdf7139fcf5d1fd7862d73117090f06c38c72cd361f18f7c2ce9c size: 3247
20
21 C:\Users\jose\Pictures\my_project_docker>docker push 1603pepe/my_project_docker-api_service:1.0
22 The push refers to repository [docker.io/1603pepe/my_project_docker-api_service]
23 76c1d0e5e28a: Pushed
24 d84f4112a51d: Pushed
25 34f7184834b2: Mounted from library/openjdk
26 5836ece05bfd: Mounted from library/openjdk
27 72e830a4dff5: Mounted from library/openjdk
28 1.0: digest: sha256:9838920329fbc685caeb2530cc1686f0bcd7b74772abfb657dcb382c044e48d7 size: 1369
29
30 C:\Users\jose\Pictures\my_project_docker>docker push 1603pepe/my_project_docker-frontend_service:1.0
31 The push refers to repository [docker.io/1603pepe/my_project_docker-frontend_service]
32 a145f2852bc1: Pushed
33 2053cbca5c04: Pushed
34 8743919be260: Pushed
35 a258a309f52e: Mounted from library/node
36 0563f6148117: Mounted from library/node
37 206e1855e6d1: Mounted from library/node
38 d4fc045c9e3a: Mounted from library/node
39 1.0: digest: sha256:b3a623eeb1ee33f74ed6286398bba09962e7cfdef529b31f7338829aab760bde size: 1785
40
41 C:\Users\jose\Pictures\my_project_docker>
```

Siguiendo los las instrucciones del paso 1, debería poderse ver las imágenes a Docker y luego, cualquier persona a Docker-Hub podrá descargar las imágenes en sus propios entornos Docker.

Paso 2: Compartir los Repositorios de Docker Hub

Una vez que las imágenes estén en Docker Hub, puedes compartir los enlaces de los repositorios con tu maestro Zacarías. Aquí dejo los links de las imágenes:

❖ Frontend Service:

https://hub.docker.com/r/1603pepe/my_project_docker-frontend_service

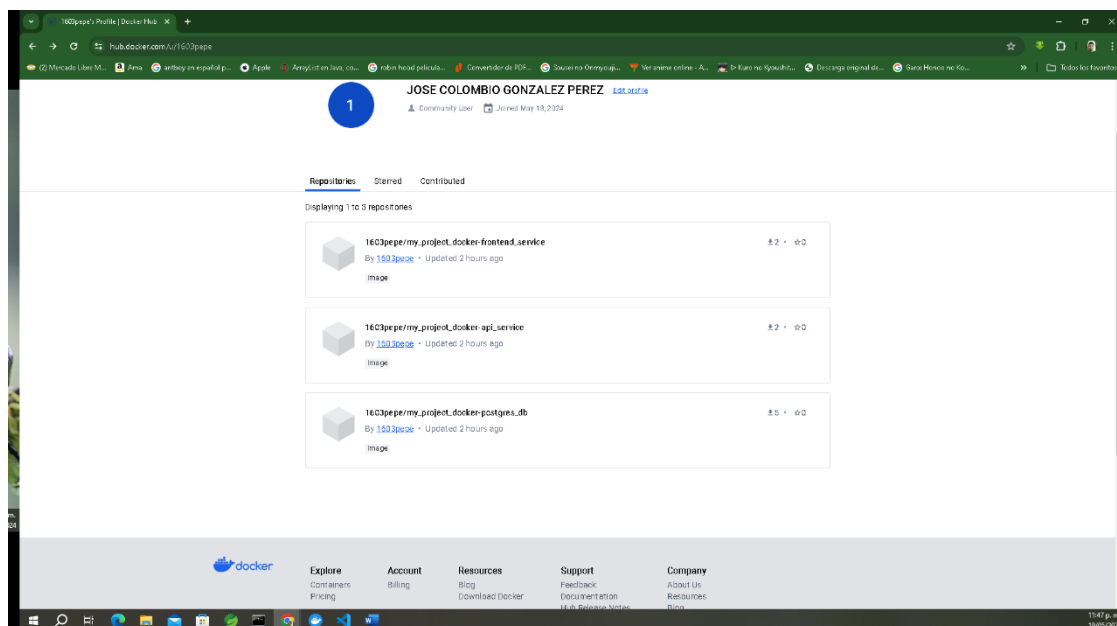
❖ API Service:

https://hub.docker.com/r/1603pepe/my_project_docker-api_service

❖ Postgres DB:

https://hub.docker.com/r/1603pepe/my_project_docker-postgres_db

Ejemplo:



Ahora qué tal si queremos ejecutar los contenedores. ¿Como lo hacemos?, pues agregamos un paso más.

Paso 3: Instrucciones para Ejecutar los Contenedores

Proporciona a tu maestro las instrucciones necesarias para ejecutar los contenedores. Aquí hay un ejemplo de lo que podría incluir:

1. Asegurarse de tener Docker y Docker Compose instalados:

- ❖ Instalar Docker
- ❖ Instalar Docker Compose

2. Instrucciones de construcción:

- ❖ Clonar tu repositorio de GitHub

Puedes hacer esto desde tu terminal de comandos con la instrucción:

Git https://github.com/pepe1603/my_project_docker.git

Ahora puedes moverte a la carpeta raíz del proyecto

cd my_project_docker

3. Instrucciones para ejecutar el proyecto:

Una vez que te encuentres dentro de la carpeta dentro del directorio raíz ejecuta el comando:

docker-compose up --build

Esto construirá en caso de no estar contruidos aun y ejecutará los servicios **postgres_db**, **api_service** y **frontend service**.

En la terminal nos aparecerá resultado que nos indican los puertos en los que están corriendo los servicios.

Acceso a los Servicios

- ❖ Frontend: <http://localhost:5175> (puedes usar el navegador)
- ❖ API: <http://localhost:9090> (puedes usar postman para test)
- ❖ Base de datos: `localhost:5432`

Con todos estos pasos definidos, deberías poder probar los contenedores y los servicios en tu máquina.

Conclusión

Mediante la configuración adecuada de Dockerfile y **docker-compose.yml**, se facilita el desarrollo, despliegue y administración de aplicaciones complejas que constan de múltiples servicios. Esto no solo asegura un entorno de ejecución consistente, sino que también simplifica el proceso de integración y despliegue continuo cuando queremos implementar más de un servicio y migrar o alojar en sistemas de alojamiento en la nube o servidores de producción sin tener ese problema clásico de ‘ **pero si en mi maquina funcionaba**’, por ende puedes Seguir esta guía, y tendrás configurados tus servicios en Docker de manera que se comuniquen correctamente entre sí usando los nombres de los servicios de Docker en lugar de **localhost**. Esto asegura que tu API se conecte a la base de datos y que tu aplicación frontend pueda comunicarse con la API sin problemas de configuración de red que implementa docker internamente.

Referencias

Docker.com. (s.f.). *docker.com*. Obtenido de
<https://www.docker.com/>

Hostinger.com. (s.f.). *Hostinger-Tutoriales*. Obtenido de
<https://www.hostinger.es/tutoriales/que-es-docker>

<https://www.docker.com/101-tutorial/>